



AFRL-RI-RS-TR-2015-006

**WIRELESS AUTHENTICATION PROTOCOL IMPLEMENTATION:  
DESCRIPTIONS OF A ZERO-KNOWLEDGE PROOF (ZKP)  
PROTOCOL IMPLEMENTATION FOR TESTING ON GROUND AND  
AIRBORNE MOBILE NETWORKS**

---

*JANUARY 2015*

TECHNICAL REPORT

*APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED*

STINFO COPY

**AIR FORCE RESEARCH LABORATORY  
INFORMATION DIRECTORATE**

## **NOTICE AND SIGNATURE PAGE**

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the 88<sup>th</sup> ABW, Wright-Patterson AFB Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (<http://www.dtic.mil>).

AFRL-RI-RS-TR-2015-006 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

**/ S /**

RICHARD MICHALAK  
Chief, Information Transmission Branch

**/ S /**

MARK H. LINDERMAN  
Technical Advisor, Computing &  
Communications Division  
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

<b>REPORT DOCUMENTATION PAGE</b>				<b>Form Approved OMB No. 0704-0188</b>	
<p>The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.</p> <p><b>PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.</b></p>					
<b>1. REPORT DATE (DD-MM-YYYY)</b> JANUARY 2015		<b>2. REPORT TYPE</b> TECHNICAL REPORT		<b>3. DATES COVERED (From - To)</b> DEC 2011 – FEB 2012	
<b>4. TITLE AND SUBTITLE</b>  WIRELESS AUTHENTICATION PROTOCOL IMPLEMENTATION: DESCRIPTIONS OF A ZERO-KNOWLEDGE PROOF (ZKP) PROTOCOL IMPLEMENTATION FOR TESTING ON GROUND AND AIRBORNE MOBILE NETWORKS				<b>5a. CONTRACT NUMBER</b> IN-HOUSE	
				<b>5b. GRANT NUMBER</b> N/A	
				<b>5c. PROGRAM ELEMENT NUMBER</b> 62702F	
<b>6. AUTHOR(S)</b>  Thomas Scatko and Nathaniel Rowe				<b>5d. PROJECT NUMBER</b>	
				<b>5e. TASK NUMBER</b>	
				<b>5f. WORK UNIT NUMBER</b>	
<b>7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)</b> Air Force Research Laboratory/RITE 525 Brooks Road Rome NY 13441-4505				<b>8. PERFORMING ORGANIZATION REPORT NUMBER</b>	
<b>9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)</b>  Air Force Research Laboratory/RITE 525 Brooks Road Rome NY 13441-4505				<b>10. SPONSOR/MONITOR'S ACRONYM(S)</b>  AFRL/RI	
				<b>11. SPONSOR/MONITOR'S REPORT NUMBER</b>  AFRL-RI-RS-TR-2015-006	
<b>12. DISTRIBUTION AVAILABILITY STATEMENT</b>  Approved for Public Release; Distribution Unlimited. PA# 88ABW-2014-5867 Date Cleared: 11 DEC 2014					
<b>13. SUPPLEMENTARY NOTES</b> This technical report presents short term in-house work performed by AFRL scientists and engineers in their official work capacity. No official in-house JON/project was established for this effort.					
<b>14. ABSTRACT</b> Authentication is deemed to be a critical function in the operation of tactical wireless ad hoc networks. The dynamic nature and unpredictability of these self-organizing networks requires that new security protocols be deployed that allow users to efficiently gain access to network resources without the burden of a centralized security infrastructure. Authentication protocols based on Zero-Knowledge Proof (ZKP) of identity schemes provide a means for establishing mutual trust between network entities. While many papers have looked at the virtues of ZKP-based authentication protocols from an academic perspective, little work has been carried out to actually deploy and test the protocols in fielded wireless networks. In this paper we present lessons-learned regarding the installation of ZKP-based authentication protocol on processing hardware designed for deployment on AFRL's small unmanned aerial vehicle (UAV) test bed.					
<b>15. SUBJECT TERMS</b>  Zero-Knowledge Proof Protocol Testing					
<b>16. SECURITY CLASSIFICATION OF:</b>			<b>17. LIMITATION OF ABSTRACT</b>	<b>18. NUMBER OF PAGES</b>	<b>19a. NAME OF RESPONSIBLE PERSON</b> THOMAS SCATKO
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U			<b>19b. TELEPHONE NUMBER (Include area code)</b> N/A

## Contents

List of Figures .....	ii
List of Tables .....	ii
1.0 Summary .....	1
2.0 Introduction .....	2
3.0 Methods, Assumptions, and Procedures .....	4
4.0 Results and Discussion .....	12
5.0 Conclusions .....	16
6.0 Recommendations .....	18
7.0 References .....	21
Appendix A: ZKP Prover Code .....	22
Appendix B: ZKP Verifier Code .....	36
List of Symbols, Abbreviations, and Acronyms .....	46

## Figures

Figure 1: Typical air-ground network scenario. ....	2
Figure 2: Typical single round authentication process. ....	5
Figure 3: Two processor benchtop wireless setup. ....	6
Figure 4: UAV payload processor device. ....	7
Figure 5: UAV ground station electronics package. ....	8
Figure 6: Benchtop test set-up. ....	10
Figure 7: Prover authentication screen display. ....	12
Figure 8: Verifier authentication screen display. ....	12
Figure 9: Prover response attribute. ....	13
Figure 10: Verifier response attributes. ....	14
Figure 11: Verifier unsuccessful authentication attempt display. ....	15
Figure 12: AFRL Stockbridge Test Site. ....	18
Figure 13: AFRL small UAVs. ....	18
Figure 14: Upgraded ground station. ....	19
Figure 15: Upgraded payload package. ....	20
Figure 16: Miniature Processor payload. ....	20

## Tables

Table 1: UAV payload computer specifications. ....	9
Table 2: Ground station computer specifications. ....	9
Table 3: UAV payload and ground station wireless modem specifications. ....	10
Table 4: Verifier response markers. ....	13
Table 5: Verifier response markers. ....	14

## 1.0 Summary

This Technical Note describes work performed to implement a ZKP-based authentication protocol on small stand-alone processors suitable for use on small UAVs and ground vehicles. The objective of this effort was to transfer the ZKP protocols from a (laboratory) desktop operating environment to a (deployable) mobile wireless set-up. The targeted systems consisted of commercial-off-the-shelf (COTS) processors and wireless networking hardware.

Two weeks of engineering/technician time was allocated to perform the required work: identify available system hardware and LINUX operating system software; configure the standalone processing nodes; install the operating systems, software drivers, and necessary math libraries; port over the ZKP authentication protocols; and finally, test and demonstrate protocol operation over wireless communication links. The resulting lessons-learned will be used in the planning of future field experiments involving (air-to-air and air-to-ground) mobile wireless networks.

The most arduous task involved configuring the processors' LINUX operating system. While nearly all of the additional software routines needed were available as *freeware* over the internet, they could not be downloaded directly to the processors through AFRL's network connections. Instead, the code had to be first downloaded to a CD-ROM then transferred to the standalone processors. Unfortunately, none of the embedded processors were configured to support CD-ROM drives i.e., size, weight and power (SWAP) requirements precluded their use on the UAV and ground stations. An inordinate amount of time was spent locating compatible CD-ROM drives and software drivers, installing them on each of the processors, and performing the needed software upgrades.

Porting of the ZKP authentication code proved to be straightforward. Only minor changes to the code were required: changing device IP addresses, and modification of certain lines of code used for the set-up of communication sockets. In addition, certain math libraries were needed to compile the code. Once again the issue of downloading code from the internet to the embedded processors had to be addressed.

Testing of the ZKP protocol, between various pairs of standalone processors, was not without problems. While the wireless networks that were implemented allowed for communications between nodes, successful operation the ZKP authentication code was achieved on only one pair of like devices i.e., a set of ground stations. Details regarding ZKP protocol implementation, testing results, and recommendations for follow-on work are documented in the remainder of this document.

## 2.0 Introduction

Authentication is required for secure interactions in tactical wireless mobile networks. However, implementing authentication protocols in self-configuring networks can be problematic given that many of today's currently deployed protocols make use of centralized security policies and controlled security infrastructure i.e., public-key encryption mechanisms and hardware to prevent unauthorized access, which demands that there exist some degree of pre-defined network structure which would, in effect, be counterproductive with regard to creating an ad hoc network.

A simple tactical ad hoc networking scenario is illustrated in Figure 1. Here, an UAV is shown entering a battlespace where it must a) establish a network connection with airborne network assets already in theater, and b) make its presence known to ground (network) users so that they can establish network connections in order to access on-board sensor resources. Here we assume that any node entering a network has no a priori knowledge of the existing network topology and has the added responsibility of discovering which nodes it can securely communicate with. Furthermore, it is assumed that in the network not all nodes are able to directly communicate with each other, in which case information may need to be relayed across the network [1]. In both situations, the amount of time available for establishing a trusted communication maybe limited. Therefore it is important that the initiation of network connectivity and access take no longer than is absolutely necessary. Having efficient security and trust management schemes is critical to providing high-speed access to military networks.

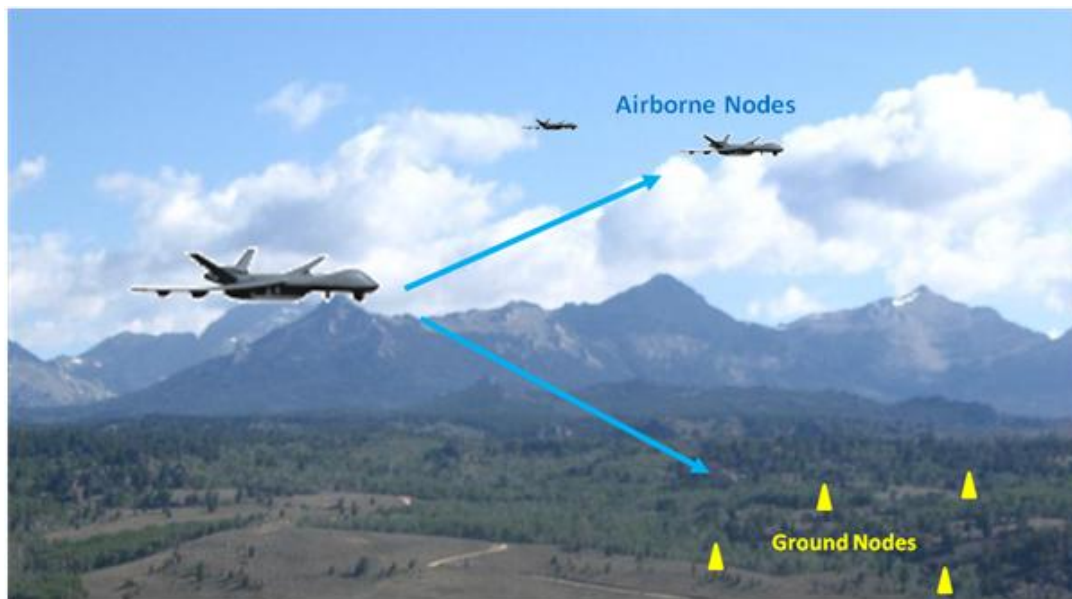


Figure 1: Typical air-ground network scenario.

Authentication of user identity must be a precondition to establishing secure communications channels in a mobile ad hoc network (MANET) [2]. Security requirements in tactical MANETS are considerably stricter than those of civilian mobile networks in that users exist at different levels of authority and therefore require different access rights and priorities. A main requirement for a authentication scheme used in MANETs are: authorized users must get access when required; the protocol used must not reduce system availability; the communicating entities must be able to verify the identity of the other party at the authority level required for the type of information exchanged; and unauthorized users must not gain access to network resources and protected data [3]. Use of ZKP-based authentication schemes due not rely on the availability of a centralized management of security credential and therefore lend themselves to use in wireless mobile networks.



### 3.0 Methods, Assumptions, and Procedures

The authentication protocol selected for use in this effort is based on the Feige-Fiat-Shamir (FFS) zero-knowledge proof (ZKP) of identity scheme [4], as implemented in the C computer language by Daniele Raffo [5], and later modified by Nathaniel Rowe [6] for use on an AFRL TCP/IP network. The C++ code for the FFS ZKP code demonstrated here is documented in Appendices A and B of this report.

The FFS ZKP identification scheme allows for anonymous authentication of identity through a series of transactions involving two entities: the *Prover*, and the *Verifier*. The basic premise underlying the scheme is that the *Prover* possess a secret token and seeks authentication by the *Verifier*, who must authenticate the *Prover* based upon the secret token that the *Prover* has, through a series of challenges without getting to know the *Prover's* secret token [7]. A simplified version of this protocol can be written [8],[9] as follows:

- An arbitrator (i.e., trusted third party) generates a random modulus  $n$  which is the product of two large Blum primes integers  $p, q$  (i.e,  $n = pq$ ). The arbitrator then publishes  $n$ .
- Using  $n$ , the *Prover* creates a secret vector  $s = [s_1, s_2, \dots, s_k]$  with  $\gcd(s_i, n) = 1$ , and  $c = \{c_{i=1,k} \mid c \in (0,1)\}$ . *Prover* next computes a public vector  $v = [v_1, v_2, \dots, v_k]$  using  $v$ , where  $v_i \equiv s_i^2 \pmod{n}$ , and publishes the result.
- The authentication protocol takes the form:
  1. *Prover* picks a random integer  $r$ , a random sign  $b \in \{-1,1\}$  and computes  $x = (-1)^{c_{i=1,k}} \cdot r^2 \pmod{n}$ , otherwise referred to as a witness, and sends it to the *Verifier*.
  2. *Verifier* selects random Booleans  $[a_1, \dots, a_i]$  where  $a_i \in \{0,1\}$ . The *Verifier* then sends this vector to the *Prover* as a challenge.
  3. *Prover* computes  $y = r (s_1^{a_1} s_2^{a_2} \dots s_k^{a_k}) \pmod{n}$  using its private key. The *Prover* then sends the result to the *Verifier*.
  4. *Verifier* computes  $z = x (v_1^{a_1} v_2^{a_2} \dots v_k^{a_k}) \pmod{n}$  and confirms  $z = y^2 \pmod{n}$ . If the relation proves true, then proof of identity is accepted. Confirmation of pass/fail is sent to the *Prover*.

The series of transactions that make up the protocol are diagrammed in Figure 2. Depending on the level of security required, the process can be repeated a number of times thereby decreasing the probability that the *Verifier* can be fooled by a dishonest or malicious *Prover*.

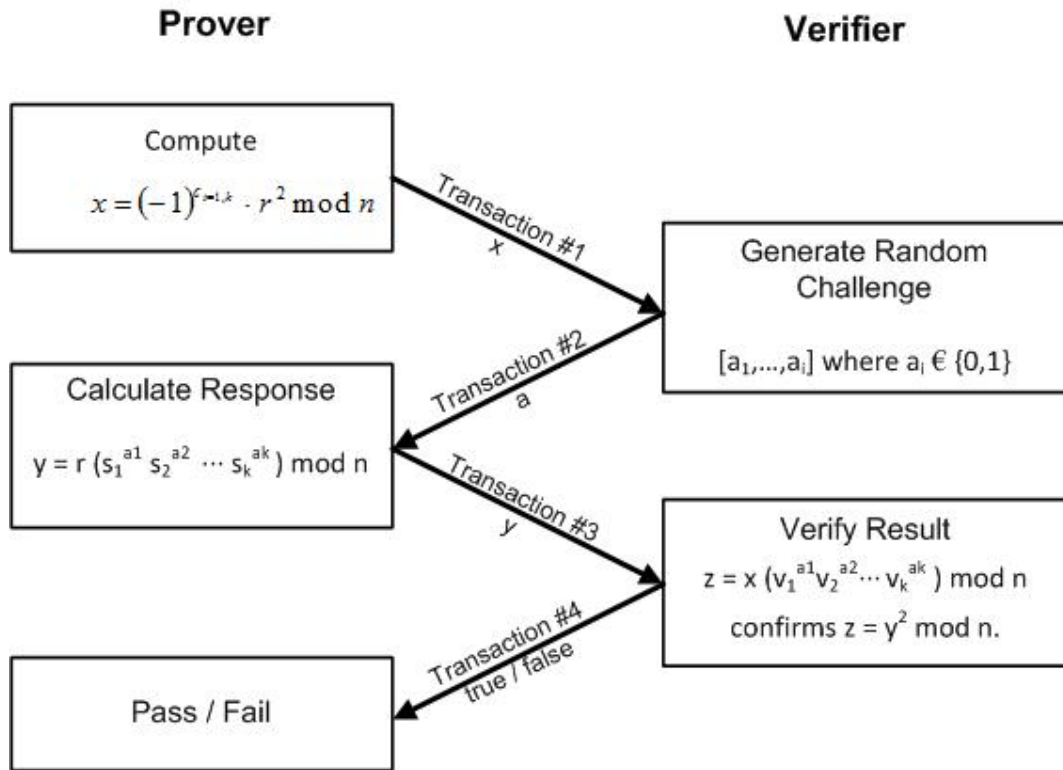
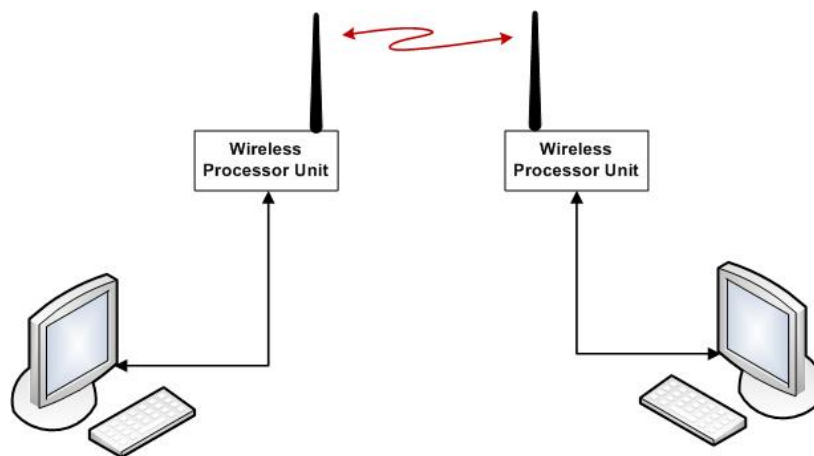


Figure 2: Typical single round authentication process.

While wireless communications frees users from having to deal with a hardwired infrastructure and fixed topology it, unfortunately, makes implementing security and trust management more difficult. Mobile wireless network operations can be impacted as users move in and out of coverage range. As the distance between wireless nodes increases, the signal to noise ratio decreases and the achievable throughput is reduced making network operations somewhat unpredictable [10]. This can challenge network activities and result in high error rate and asymmetric data rates, low transmission rates, large latencies, and intermittent connectivity, as well as disconnection, and intermittent and unpredictable operation [11]. Additionally, efficient authentication protocol operation can also be affected due to the unavailability of resources. Deployed devices may have limited amounts of processing capability, memory, and power availability which may delay, or restrict, the time exchange of data. As a result, the testing of authentication protocols need to be carried out using systems appropriate to mobile applications rather than on hard-wired, laboratory computer networks.

Our approach to testing involved porting existing ZKP-based authentication protocols (which have been tested on networked desktop computers) to field-deployable, battery-operated wireless devices destined for use in AFRL's in-house small UAV research program. A basic two-node wireless network was set-up in an indoor laboratory to allow for experimentation with various mobile device configurations to demonstrate operation of the authentication protocol. The block diagram provided in Figure 3 illustrates the configuration of the test network. Results obtained from this initial round of testing would help determine the course of future outdoor experiments involving mobile airborne and mobile ground nodes.



**Figure 3: Two processor benchtop wireless setup.**

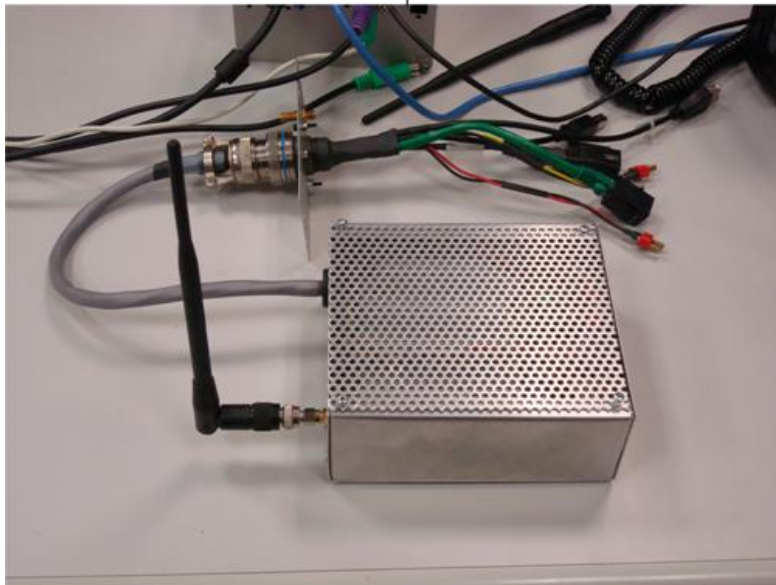
Two different wireless computer systems were selected: a payload system designed for installation on a small UAV; and a portable ground station used for communicating with the airborne payload, but also can be configured for mobile ground applications. The UAV payload electronics package is shown in Figure 4 while the ground station is illustrated in Figure 5. Both the payload and ground stations systems are designed to operate off of external (battery) power sources that are

installed on either the UAV or the mobile ground vehicles. For benchtop testing purposes commercial dc power supplies were employed. In addition, all systems tested were outfitted with 5.5 dBi *rubber duck* antennas. The rubber duck antenna screws directly into the existing external antenna port and swivels 360 degrees as well as tilts up to 90 degrees.

The UAV payload package consists of a processor board (model: PM-945GSE-270) and wireless radio board (model: TL-WN861N) which are stacked and attached to a convection-cooled heatsink. An instrumentation cable allows for external connections to a monitor, keyboard and network for performing preflight tests.



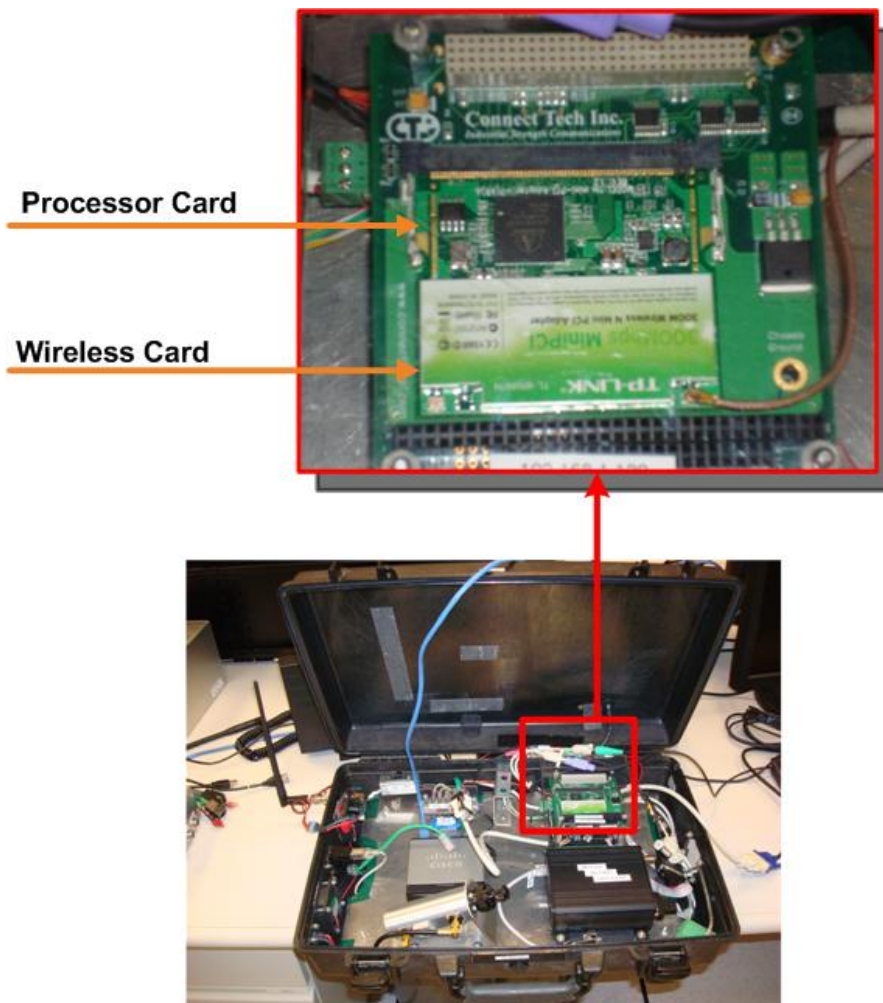
**A) Electronics**



**B) Payload**

**Figure 4: UAV payload processor device.**

The ground station is outfitted with a number of devices which include a processor board (model: PM-LX2-800-R10) and wireless radio board (model: TL-WN861N) along with an RF attenuator, a Cisco® five port 10/100 switch, and a Microhard Systems® spread spectrum modem (model: MHX320). For the tests described herein only the processor and RF radio boards were utilized.



**Figure 5: UAV ground station electronics package.**

The PM-945GSE-270 and PM-LX2-800-R10 are highly integrated embedded computers specifically optimized for multi-media applications requiring minimum installation space [12]. The processor board is particularly suitable for low power and fan-less applications. Specifications for these single-board computers are provided in Tables 1 and 2. Both systems use the same embedded radio frequency (RF) wireless radio: model TL-WN861N. Specifications for the RF wireless board [13] are listed in Table 3.

**Table 1: UAV payload computer specifications.**

Specification/Model	PM-945GSE-N270
Form Factor	PCI-104 Module
Processor	Intel® Atom™ N270 CPU
Integrated Graphics	Intel®945GSE
On-board Static Memory	1 GByte DDR2 SDRAM
System Controller Hub Chipset	Intel®ICH7M
BIOS	AMI BIOS
Compatible OS	Microsoft® Windows XP™ SP2 Microsoft® Windows Vista Business™ (32bit) Linux Ubuntu™ 8.10 Linux Fedora™ Core 6
Ethernet Controller	Realtek® RTL8102E
Super I/O Controller	Fintek® F81865
Dimensions	108.6 mm x 115.6 mm
Weight	250 g
Power consumption	5 VDC @ 2.6 A

**Table 2: Ground station computer specifications.**

Specification/Model	PM-LX2-800-R10
Form Factor	PC-104 Module
Processor	AMD® Geode™ LX800 CPU
Integrated Graphics	AMD® LX800
On-board Static Memory	1 GByte DDR2 SDRAM
System Controller Hub Chipset	SMSC 3114
BIOS	AMI BIOS
Compatible OS	Microsoft® Windows XP™ SP2 Microsoft® Windows Vista Business™ (32bit) Linux Ubuntu™ 8.10 Linux Fedora™ Core 6
Ethernet Controller	Realtek® RTL8100C
Super I/O Controller	SMSC 3114
Dimensions	90 mm x 90 mm
Weight	110 g
Power consumption	5 VDC @ 1.09 A

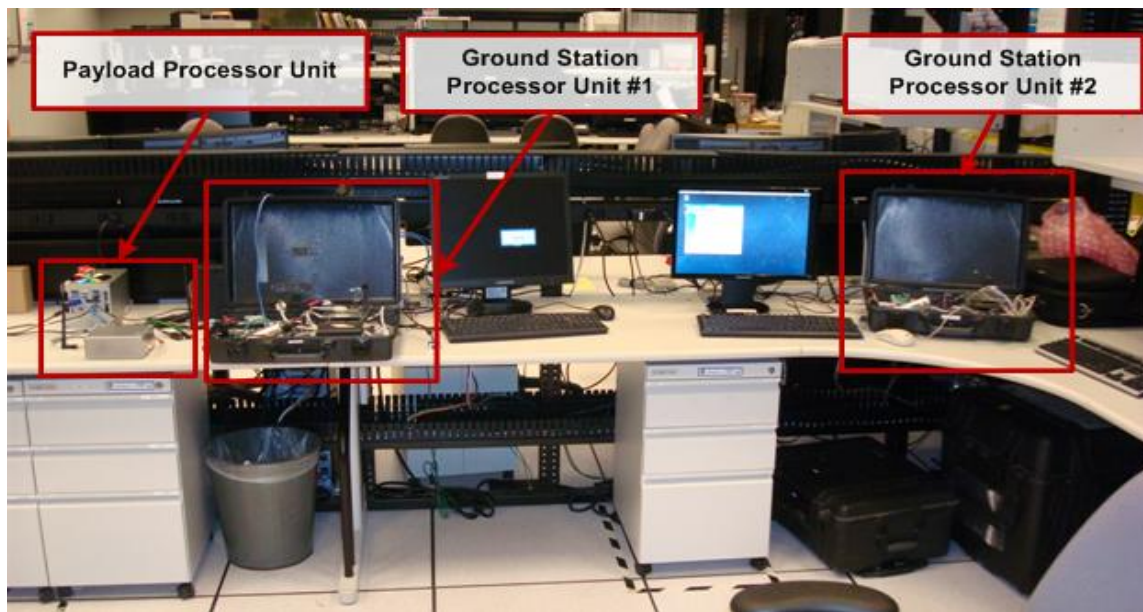
The TL-WN861N wireless adapter board complies with IEEE 802.11n, IEEE 802.11g, and IEEE 802.11b standards. Wireless transmission rates are specified up to 300Mbps. This device supports 64/128/152-bit WEP, WPA/WPA2 and WPA-PSK/WPA2-PSK encryptions. This device can also simultaneously operate bandwidth intensive applications such as voice and video. Applications requiring a lot of bandwidth and that are sensitive to interruptions, such as voice and video applications, are given priority in order to assure quality. The manufacturer claims that it also works well with other 11g and 11n protocol wireless products.



**Table 3: UAV payload and ground station wireless modem specifications.**

Specification/Model	TL-WN861N
Form Factor	32-bit Mini PCI
Standards	IEEE 802.11n/g/b CSMA/CA with ACK
Wireless Signal Rates With Automatic Fallback	11n:270/243/216/162/108/81/54/27 Mbps 135/21.5/108/81/54/40.5/27/13.5 Mbps 130/117/104/78/52/39/26/13 Mbps 65/58.5/52/39/26/19.5/13/6.5Mbps 11g:54/48/36/24/18/12/9/6M (adaptive) 11b:11/5.5/2/1M (adaptive)
Frequency Range	2.4-2.4835 GHz
Wireless Transmit Power	20 dBm (max)
Modulation Type	OFDM/CCK/16-QAM/64-QAM
Receiver Sensitivity	270 m -68 dBm@10%PER 130 m -68 dBm@10%PER 108 m -68 dBm@10%PER 54 m -68 dBm@10%PER 11 m -85 dBm@8%PER 6 m -68 dBm@10%PER 1 m -90 dBm@10%PER
Security	64/128/152 bit WEP, WPA/WPA/WPA2, WPA-PSK/WPA2-PSK (TKIP/AES)

The laboratory benchtop set-up for testing the authentication protocols between the two different types of wireless systems is shown in Figure 6.



**Figure 6: Benchtop test set-up.**

Two different system implementations of the authentication protocol were tested – with network configurations limited to just two nodes. One case involved the use of two ground station units as might be deployed in a mobile ground network. The other case employed a ground station and a UAV payload to represent deployment in an air-to-ground network. In both cases, *Prover* code was installed on one of the nodes, while *Verifier* code was installed on the other. As a follow-on test, installation of the *Prover* and *Verifier* code was switched between the nodes in order to demonstrate that a node could function as either a *Prover* or *Verifier*. Results of the tests run are discussed in the section that follows.

Prior to protocol testing, the configuration of the LINUX operating systems were verified. Already installed on each system was LINUX Fedora™ release-11 and only minor software updates had to be made. Compiling the ZKP authentication code did require use of GMP math library functions. GMP is a free library, available for download via the internet, for arbitrary precision arithmetic, operating on signed integers, rational numbers and floating point numbers. The main target applications for GMP are cryptographic applications and research, internet security and algebraic systems and research [14]. Installing the GMP libraries on the individual systems proved to be problem.

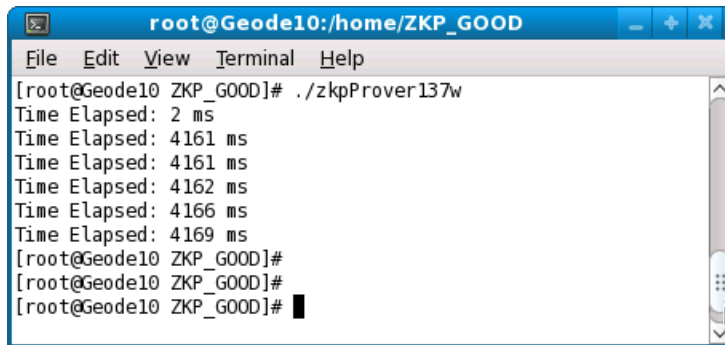
AFRL policies do not allow for unsecured systems to be connected to the internet so direct download of the GMP files was not a possibility. While these files could be download to a CD-Rom via an authorized desktop computer, neither the UAV payload nor the Ground Station Units were configured to support installation of a CD-Rom drive. In the end, it was decided to download the GMP files to a stand-alone computer system, compile the C code, then download the compiled code to the UAV payload and Ground Station processors using a wired Ethernet connection. This hard-wired connection was disconnected prior to wireless testing.



## 4.0 Results and Discussion

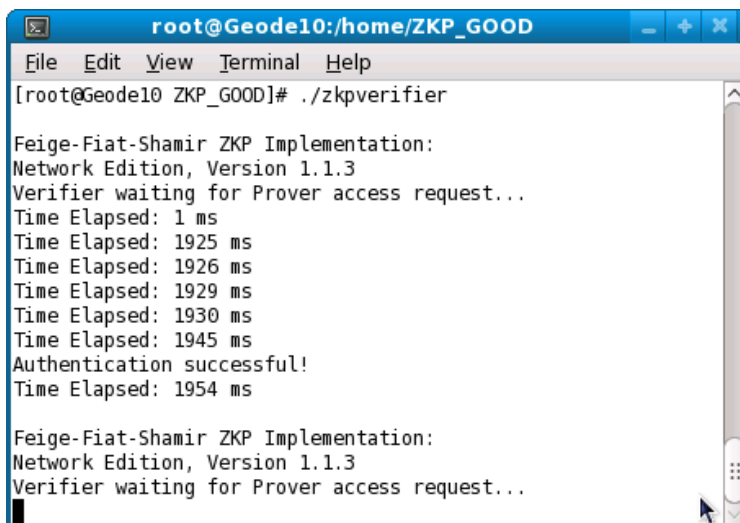
The objective of this short-duration effort involved porting a ZKP-base authentication protocol from a desktop, cabled network set-up to a mobile, wireless network configuration for the purpose of demonstrating operation over a wireless communication link. The expectations were that a successful outcome would provide impetus for developing a more detailed plan for field deployment of the protocol using mobile (air and ground) test assets.

The authentication protocol consists of a series of exchanges between two entities: a *Prover*, and a *Verifier* (as was described in Section 3 and illustrated in Figure 2). The exchanges that occur are made observable through use of timestamps that are output to display consoles. For the test case involving the two ground stations, the resulting exchange reports for the *Prover* and *Verifier* are shown in Figures 7 and 8 respectively.



```
root@Geode10:/home/ZKP_GOOD
File Edit View Terminal Help
[root@Geode10 ZKP_GOOD]# ./zkpProver137w
Time Elapsed: 2 ms
Time Elapsed: 4161 ms
Time Elapsed: 4161 ms
Time Elapsed: 4162 ms
Time Elapsed: 4166 ms
Time Elapsed: 4169 ms
[root@Geode10 ZKP_GOOD]#
[root@Geode10 ZKP_GOOD]#
[root@Geode10 ZKP_GOOD]#
```

Figure 7: Prover authentication screen display.



```
root@Geode10:/home/ZKP_GOOD
File Edit View Terminal Help
[root@Geode10 ZKP_GOOD]# ./zkpverifier

Feige-Fiat-Shamir ZKP Implementation:
Network Edition, Version 1.1.3
Verifier waiting for Prover access request...
Time Elapsed: 1 ms
Time Elapsed: 1925 ms
Time Elapsed: 1926 ms
Time Elapsed: 1929 ms
Time Elapsed: 1930 ms
Time Elapsed: 1945 ms
Authentication successful!
Time Elapsed: 1954 ms

Feige-Fiat-Shamir ZKP Implementation:
Network Edition, Version 1.1.3
Verifier waiting for Prover access request...
█
```

Figure 8: Verifier authentication screen display.

Figure 9 and Table 4 identify and describe the timestamps generated by the *Prover's* code, while Figure 10 and Table 5 describe the timestamps output by the *Verifier's* code.

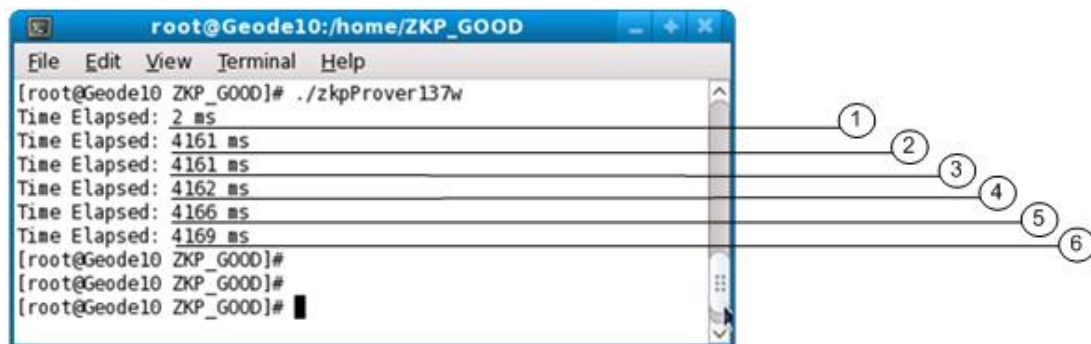


Figure 9: Prover response attribute.

Table 4: Verifier response timestamp markers.

Marker	Description
1	1 <sup>st</sup> Timestamp: <i>Prover</i> establish communications connection
2	2 <sup>nd</sup> Timestamp: <i>Prover</i> sends/publishes <i>Public Key</i>
3	3 <sup>rd</sup> Timestamp: <i>Prover</i> sends/publishes <i>modulos n</i>
4	4 <sup>th</sup> Timestamp: <i>Prover</i> sends <i>witness (x)</i>
5	5 <sup>th</sup> Timestamp: <i>Prover</i> accepts <i>challenge (a)</i>
6	6 <sup>th</sup> Timestamp: <i>Prover</i> sends <i>response (y)</i>

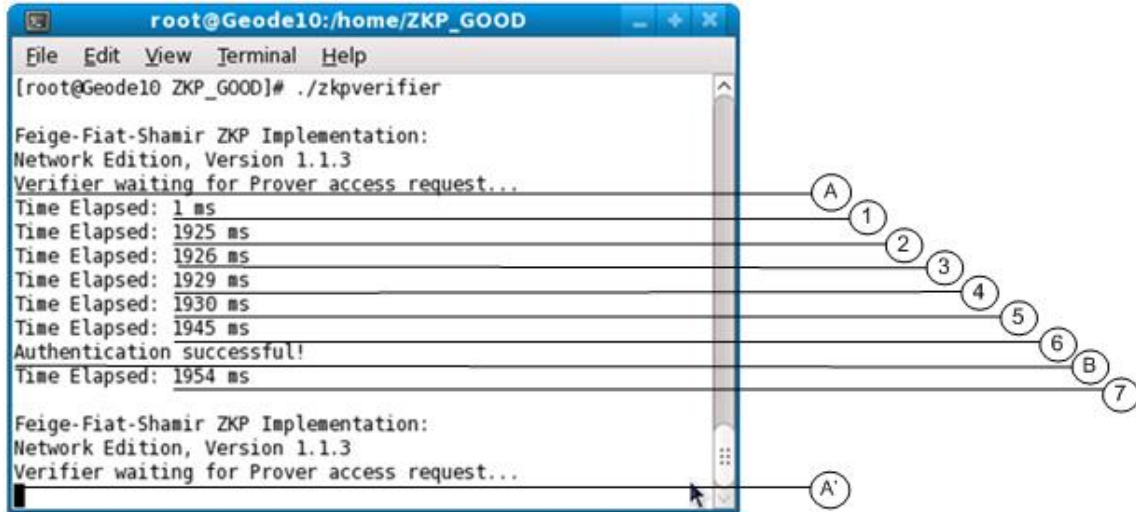
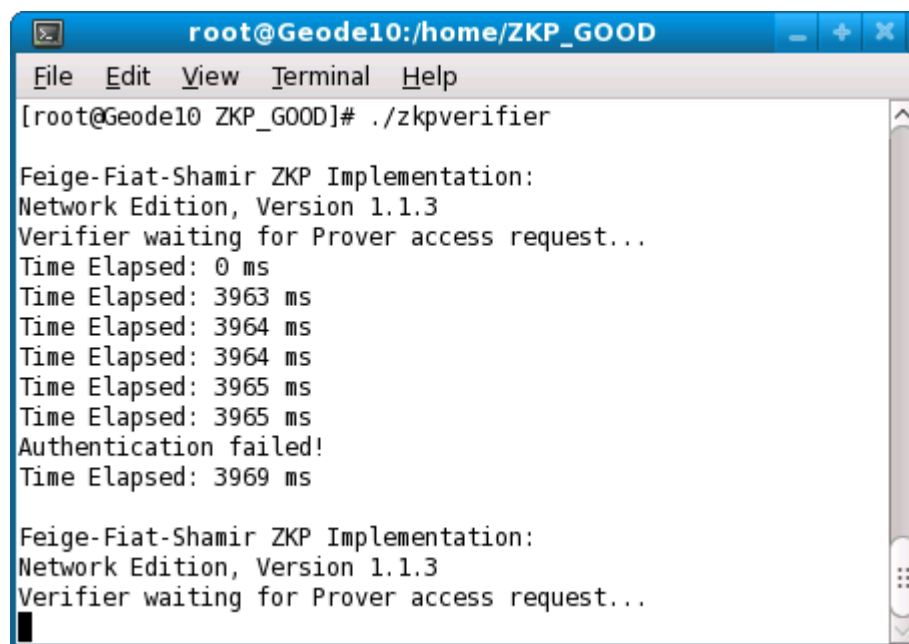


Figure 10: Verifier response attributes.

Table 5: Verifier response timestamp markers.

Marker	Description
A	Machine is awaiting receipt of a request
1	1 <sup>st</sup> Timestamp: Start counter upon receipt of a request
2	2 <sup>nd</sup> Timestamp: Verifier receives <i>public key</i>
3	3 <sup>rd</sup> Timestamp: Verifier receives <i>modulos n</i>
4	4 <sup>th</sup> Timestamp: Verifier receives <i>witness (x)</i>
5	5 <sup>th</sup> Timestamp: Verifier sends <i>challenge (a)</i>
6	6 <sup>th</sup> Timestamp: Verifier receives <i>response (y)</i>
B	Verify Authenticity of Prover
7	7 <sup>th</sup> Timestamp: Elapsed time for completing authentication protocol
A'	Machine is awaiting receipt of a request

An example screen display of a failed authentication attempt is provided in Figure 11. In this case, the *Prover-Verifier* configuration was identical to that described above i.e., two ground station systems communication over a wireless link. The failure was ultimately found to be caused by having the ground station units' processor board's Ethernet connection enabled at the time that the RF radio's wireless connection was enabled. A hardwired Ethernet connection was made between the two units for downloading compiled code and system configuration checking. Disabling the Ethernet connection via an operating system call solved the protocol failure problem. It is not entirely clear why having both communication channels enabled should have caused a failure.

A screenshot of a terminal window titled 'root@Geode10:/home/ZKP\_GOOD'. The window has a menu bar with 'File', 'Edit', 'View', 'Terminal', and 'Help'. The terminal text shows the execution of './zkpverifier'. The output includes: 'Feige-Fiat-Shamir ZKP Implementation:', 'Network Edition, Version 1.1.3', 'Verifier waiting for Prover access request...', 'Time Elapsed: 0 ms', 'Time Elapsed: 3963 ms', 'Time Elapsed: 3964 ms', 'Time Elapsed: 3964 ms', 'Time Elapsed: 3965 ms', 'Time Elapsed: 3965 ms', 'Authentication failed!', and 'Time Elapsed: 3969 ms'. Below this, the same header information is repeated, followed by 'Verifier waiting for Prover access request...'. The cursor is at the end of the second line of output.

```
root@Geode10:/home/ZKP_GOOD
File Edit View Terminal Help
[root@Geode10 ZKP_GOOD]# ./zkpverifier

Feige-Fiat-Shamir ZKP Implementation:
Network Edition, Version 1.1.3
Verifier waiting for Prover access request...
Time Elapsed: 0 ms
Time Elapsed: 3963 ms
Time Elapsed: 3964 ms
Time Elapsed: 3964 ms
Time Elapsed: 3965 ms
Time Elapsed: 3965 ms
Authentication failed!
Time Elapsed: 3969 ms

Feige-Fiat-Shamir ZKP Implementation:
Network Edition, Version 1.1.3
Verifier waiting for Prover access request...
█
```

Figure 11: Verifier unsuccessful authentication attempt display.

No results are available that show successful demonstration of protocol operation between a UAV payload system and ground station. The two-node configuration only worked using a cabled Ethernet connection with *Prover* code running on the UAV payload's processor. When *Verifier* code was run on the UAV payload, with *Prover* code running on the ground station, the protocol failed. Furthermore, when the wireless link was employed the demonstration failed regardless of whether *Prover* or *Verifier* code was being run on either of the two systems. A check was made to insure that the Ethernet communication path was disabled and to eliminate it as the reason for failure. More time would be required to identify why use of the payload system caused such problems.

## 5.0 Conclusions

The primary objective of this work was to port a ZKP authentication protocol, known to work on a desktop computer network, onto selected embedded processors and demonstrate operations over a wireless communications link. Given the short duration of the effort, it was not possible to complete an exhaustive performance evaluation. Suffice it to say, partial success was achieved in that protocol operations were fully demonstrated on a pair of Ground Station systems. Unfortunately, problems were encountered when using a UAV-Ground Station pair. With only a limited amount of in-house resources available to carry out this work, no further investigations could be carried out to determine the reason for protocol inoperability. Program results and lessons-learned are summarized below.

- The selected authentication protocol was demonstrated on two wireless ground station nodes, each having identical processing and wireless radio hardware. Also demonstrated was the ability for each node to function as both a protocol *Prover* and *Verifier*. It is noted that the protocol must be initiated manually at each of the two nodes. Additional code would be needed to automate the process.
- During testing with the two ground station nodes it was noted that the protocol would not operate over the wireless link if the on-board Ethernet connection was enabled. Disabling the Ethernet connection via an operating system call eliminated this problem. Originally, the Ethernet connection was used to down-load the compiled C++ authentication protocol code, then disconnected following completion system verification procedures and commencement of wireless testing. The Ethernet connection is made by means of a connection on the PC-104 processor board, while the wireless connection is made via the RF radio daughter-board; two separate connectors on two different boards. Further investigation of system hardware and software configurations is warranted.
- Wireless operation of the protocol between a ground station node and airborne payload node failed to be demonstrated. Bi-directional wireless communication was established between the two nodes as evidenced by the ability of each node to “ping” the other. As was previously noted, the UAV payload and ground station were not implemented using identical hardware. The pinging procedure was implemented outside of the authentication protocol code which could imply that there may be something unique about the authentication code that prevents its operation on the payload system hardware – perhaps, the manner in which the communications sockets were coded. Further investigation of this problem is warranted.
- Furthermore, testing of the protocol between the UAV payload and ground station nodes using a wired Ethernet connection was only partially successful. The payload system could only be configured to function as a *Prover* and not as a *Verifier* over the hard-wired connection. This problem may be due to how the communication sockets are set up or, perhaps, has to do with

selection of the wireless versus wired communication ports. Further investigation of this problem is warranted.

- Even though all the nodes were using the same version LINUX operating systems, there may be some features that are impacted by node system hardware and (driver) software configurations. A better understanding of the nuances regarding specific system implementations may be required, especially if the authentication protocol code is expected to be run on a variety of processing platforms. Also, a better understanding of the integration of the processing and RF radio boards may be required. This familiarity can only be gained through working with the various devices.

## 6.0 Recommendations

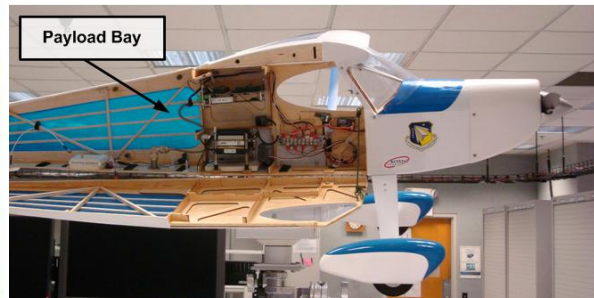
AFRL Rome Research Site technical personnel are currently pursuing technology investigations involving the use of small, low cost UAV technology. Field testing of novel COTS-based UAV networking approaches is carried out at AFRL's Stockbridge Test Site. Here, data collections are carried out to evaluate link performance, develop network protocols and applications, as well as evaluate their performance, in realistic environments [15]. The availability of the UAVs and the Stockbridge Test Site provide an ideal location for deploying and test ZKP-based authentication protocols in air-to-air, air-to-ground and mobile on-the-ground scenarios. An aerial view of the Test Site, showing the air strip used for testing the small UAVs, is provided in Figure 12. Two photographs showing typical UAV aircraft, one in the field while and another opened-up in the laboratory, are shown in Figure 13.



Figure 12: AFRL Stockbridge Test Site.



A) Small UAV at test site.



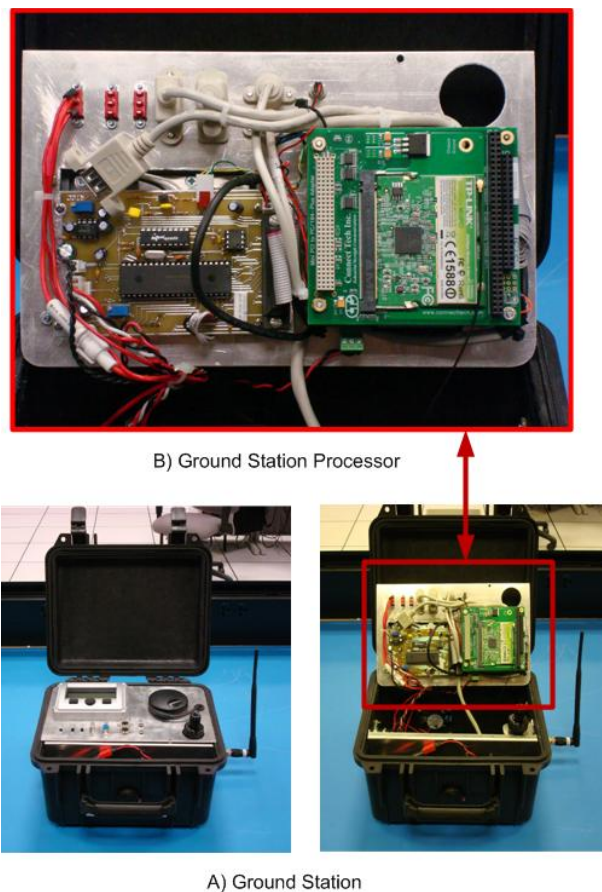
B) Small UAV payload bay.

Figure 13: AFRL small UAVs.



Porting of the ZKP authentication protocols from a hardwired desktop-centric processing environment to a wireless mobile environment demonstrated that the protocols can be easily migrated to systems designed and built for use with small UAVs. Furthermore, by making use of existing UAV test program assets the cost and risk of deploying a working wireless data link between a UAV and mobile ground station can be minimized. As has been noted earlier, some problems have been encountered when attempting to operate the ZKP protocol between systems employing different CPU boards. This problem is considered to be minor a minor one and, with some small amount of additional effort, should be easily corrected. If follow-on protocol development work will be directed towards deployment on AFRL's UAV platforms then the work needs to be coordinated with processor payload designs being developed for use under the small UAV program.

AFRL technical personnel are currently assembling new Ground Station and UAV payload systems. Next generation ground station designs make use of the same processor boards and upgraded wireless radio boards. An example of one such unit is shown in Figure 14. Given the noted difficulties in operating the authentication protocols on different processing and wireless system configurations, future protocol integration work should be carried out on the physical hardware that will be deployed in the field. Therefore, protocol development should be worked in parallel with UAV program UAV payload and ground station design and development.



**Figure 14: Upgraded ground station.**



UAV payload designs have not progressed to the point where a decision can be made regarding the selection of hardware. Two candidate platforms have been procured for evaluation purposes: Via Technologies® ARTiGo A1150™, which includes a full 64-bit dual core processor and support Wi-Fi communications; and Embedded Systems TS-4800™ controlled board, featuring an 800 MHz ARM™ processor and capable accepting a Wi-Fi capable wireless daughter board. The two devices are shown in Figures 15 and 16 respectively.



Figure 15: Upgraded payload package.

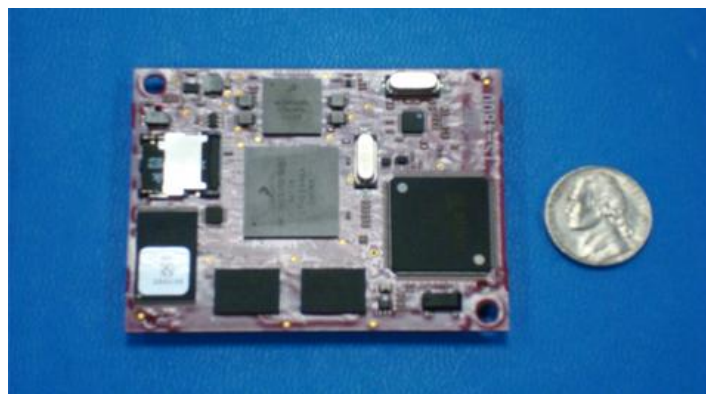


Figure 16: Miniature Processor payload.

Even though all of the above mentioned systems are capable of running LINUX operating systems, the configuration of software drivers and hardware devices may impact the operation of the authentication protocols. It would be beneficial to evaluate both the operating systems, hardware devices and authentication protocol, simultaneously, in order minimize integration problems. A more detailed test plan should be developed to guide future deployment of ZKP-based authentication protocols.

## 7.0 References

- [1] Liu, L. "Security and Trust Management in Self-organizing Wireless Ad hoc Networks", AFRL Workshop Paper, College of Computing, Georgia Institute of Technology. 2009.
- [2] Aboudagga, N., Rafaei, M., Eltoweissy, M., DaSilva L., and Quisquater, J., "Authentication Protocols for Ad Hoc Networks: Taxonomy and Research Issues", [Q2SWinet '05](#) Proceedings of the 1st ACM international workshop on Quality of service & security in wireless and mobile networks, p. 1, 2005.
- [3] Hegland, A., Winjum, E., Hedenstad, O., "A Framework for Authentication in NBD Tactical Ad Hoc Networks", IEEE Communications Magazine, October 2011, p. 64.
- [4] Feige, U., Fiat, A., Shamir, A., "Zero-Knowledge Poofs of Identity", Journal of Cryptology, 1988, p.77-94.
- [5] Raffo, D., "Digital Certificates and the Feige-Fiat-Shamir Zero-Knowledge Protocol", Labratoire d'Informatique, Ecole Polytechnique, France, pp. 37-56, 2002.
- [6] Rowe, N., unpublished work regarding implementation of FFS ZKP protocols on TCP/IP networks, 2009.
- [7] Kizza, J.M., "Feige-Fiat-Shamir ZKP Scheme Revisited", International Journal of Computing and ICT Research, Vol. 4, No. 1, p. 9, June 2010.
- [8] AFRL-RI-TR-2011-178, "Airborne Network Trust Using Zero Knowledge Techniques", Northrop Grumman Systems Co., Reily, M., Card, S., p. 61.
- [9] Aronsson, H. A., "Zero Knowledge Protocols and Small Systems", Department of Computer Science, Helsinki University of Technology, <http://www.tml.tkk.fi/Opinnot/Tik-110.501/1995/zeroknowledge.html> .
- [10] Liu, L., *ibid*, p2.
- [11] Aidi, L. Changsu, J. "Delay Tolerant Network", School of Information and Communication Technology KTH, Stockholm, Sweden, 2011, <http://www.slideshare.net/lailiaidi/delay-tolerant-network-11484982> .
- [12] PM-945GSE-N270 Users Manual, IEI Technology Corporation, 17 August 2011.
- [13] TL-WN861N Data Sheet, TP-LINK®, [www.tp-link.com](http://www.tp-link.com) .
- [14] The GNU Multiple Precision Arithmetic Library, <http://gmplib.org/> .
- [15] Hague, D., Kung, H.T., Suter, B., "Field Experimentation of COTS-Based UAV Networking", IEEE MILCOM 2006.

## **Appendix A: ZKP Prover Code**

## ***ZKP\_PROVER/Makefile***

# Makefile for the FFS Zero-Knowledge Identification Scheme implementation

CC = gcc

LIBS = /usr/local/lib/libgmp.a

zkp: zkp.o center.o prover.o

\$(CC) zkp.c center.c prover.c -o zkp -lm \$(LIBS)

clean:

clear; rm -f zkp \*~ \*.o core

## **ZKP\_PROVER/zkp.h**

```
#define TRUE 1
#define FALSE 0

/* Multiplicity of challenge */
#define K 10
/* Number of rounds of the protocol */
#define T 1

mpz_t n;          /* modulus (a Blum integer) */
mpz_t i[K];       /* Prover's public key */
mpz_t rndseed;

void setrndseed();
void publish_modulus();
void compute_keys();
```

## **ZKP\_PROVER/center.c**

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <gmp.h>
#include "zkp.h"

/*
   Publish the modulus (a Blum integer which prime factors
   are randomly chosen and 512 bits long)
*/

void publish_modulus() {
    mpz_t rand, tmpprime, tmp, prime1, prime2;
    gmp_randstate_t state;

    mpz_init(rand);
    mpz_init(tmpprime);
    mpz_init(tmp);
    mpz_init(prime1);
    mpz_init(prime2);
    mpz_init(n);

    gmp_randinit_lc_2exp_size(state, 128);

    /* computes 1st prime */
    setrandseed();
    gmp_randseed(state, rndseed);
    mpz_rrandomb(rand, state, 512);
    while (1) {
        /* repeat until prime is of
        form 4r+3 */
        mpz_nextprime(tmpprime, rand);
        mpz_sub_ui(tmp, tmpprime, 3);
        if (mpz_divisible_ui_p(tmp, 4)) break;
        mpz_set(rand, tmpprime);
    }
    mpz_set(prime1, tmpprime);

    /* computes 2nd prime */
    setrandseed();
    gmp_randseed(state, rndseed);
    mpz_rrandomb(rand, state, 512);
    while (1) {
        mpz_nextprime(tmpprime, rand);
        mpz_sub_ui(tmp, tmpprime, 3);
        if (mpz_divisible_ui_p(tmp, 4)) break;
        mpz_set(rand, tmpprime);
    }
    mpz_set(prime2, tmpprime);
```

```

/* computes modulus */
mpz_mul(n, prime1, prime2);
//gmp_printf("Publishing modulus: %Zd\n\n", n);

//mpz_clear(rand);
//mpz_clear(tmpprime);
//mpz_clear(tmp);
//mpz_clear(prime1);
//mpz_clear(prime2);
//gmp_randclear(state);
}

```

## **ZKP\_PROVER/prover.c**

```
#include <stdlib.h>
#include <stdio.h>
#include <gmp.h>
#include "zkp.h"

static mpz_t s[K];    /* private key */
static mpz_t r;       /* random number */

/*
  Choose private and public key
*/
void compute_keys() {
    int index = 0, index2, flag;
    mpz_t candidate, inverse;
    gmp_randstate_t state;

    //printf("Computing keys ");

    gmp_randinit_lc_2exp_size(state, 128);
    setrandseed();
    gmp_randseed(state, rndseed);
    mpz_init(candidate);
    mpz_init(inverse);

    while (index < K) {
        //printf(". ");
        mpz_init(i[index]);
        mpz_init(s[index]);

        mpz_urandomm(candidate, state, n);

        /* test if candidate has already been chosen as key component */
        flag = FALSE;
        for (index2 = index - 1; index2 >= 0; index2--) {
            if (mpz_cmp(s[index2], candidate) == 0) {
                flag = TRUE;
                break;
            }
        }
        if (flag == TRUE) continue;

        mpz_mul(inverse, candidate, candidate);
        mpz_mod(inverse, inverse, n);
        if (mpz_invert(inverse, inverse, n) == 0) continue;

        mpz_set(s[index], candidate);
        mpz_set(i[index], inverse);
    }
}
```



```

        index++;
    }

    //printf("\n\nPublic key:\n");
    // for (index = 0; index < K; index++)
    //{
    //gmp_printf("%Zd\n", i[index]);
    //printf("size of i: %d", (sizeof(i)/sizeof(i[0])));
    //}
    //printf("Private key:\n");
    //for (index = 0; index < K; index++) gmp_printf("%Zd\n", s[index]);

    ////////////////////////////////////// run at program close to
    clear:
    //mpz_clear(candidate);
    //mpz_clear(inverse);
    //gmp_randclear(state);
    }

/*
    Pick a random number and send the witness x (step 1 of the protocol)
*/
void witness(mpz_t x) {
    gmp_randstate_t state;

    mpz_init(r);
    mpz_init(x);

    gmp_randinit_lc_2exp_size(state, 128);
    setrandseed();
    gmp_randseed(state, rndseed);

    mpz_urandomm(r, state, n);
    mpz_mul(x, r, r);
    mpz_mod(x, x, n);
    //gmp_printf("\nWitness      : %Zd\n\n", x);

    //gmp_randclear(state);

    //return x;
}

/*
    Send the response y (step 3 of the protocol)
*/
void response(mpz_t y, unsigned int e)
{
    int index;

```

```

mpz_set(y, r);

for (index = 0; index < K; index++)
{
    if (e & (0x1 << index))
        mpz_mul(y, y, s[index]);
}
mpz_mod(y, y, n);

//gmp_printf("Response      : %Zd\n\n", y);
}

```

## **ZKP\_PROVER/zkp.c**

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <gmp.h>
#include <time.h>
#include <sys/time.h>
#include "zkp.h"

//network includes
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

/*****
*****
Feige-Fiat-Shamir (FFS) Zero Knowledge (ZK) Identification Scheme:
Networked Implementation:

Original by Daniele Raffo, 25 JUN 2002 - LIX, Ecole
Polytechnique
Expanded by Nathaniel Rowe, 11 JAN 2012 - AFRL Rome

This networked implementation uses the same FFS ZK format, but is
setup
to function over a TCP/IP based network. ZK is preserved.

This program is the ***CLIENT / PROVER*** edition

*****/

uint32_t stampstart();
uint32_t stampstop(uint32_t start);

int fastseed = TRUE;

int main(int argc, char **argv) {

    size_t words;
    int proof, temp;
    unsigned int e;          /* random boolean vector (challenge) */
    mpz_t x;                 /* witness */
    mpz_t y;                 /* response */
    uint32_t start, stop;     //time stamp start and stop

    mpz_init(x);
```

```

mpz_init(y);
mpz_init(rndseed);

//START TIME TIMESTAMP
start = stampstart();

/***** START NETWORK SETUP *****/
int sockfd, portno, s;
struct sockaddr_in serv_addr;

portno = 12303;
sockfd = socket(AF_INET, SOCK_STREAM, 0);

serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
serv_addr.sin_port = htons(portno);

s = connect(sockfd, &serv_addr, sizeof(serv_addr));
if(s<0)
    printf("Prover: Connection Failed\n");
/***** END NETWORK SETUP *****/

stop = stampstop(start);

int t_var = T;

for(t_var; t_var > 0; t_var--) //START FOR LOOP FOR T
{
    mpz_init(rndseed);
    mpz_init(y);
    mpz_init(x);

    publish_modulus();          /* Prover publishes modulus n
*/
    compute_keys();             /* Prover chooses public/private keys
*/
    witness(x);                 /* Prover sets x */

/***** START Send Public Key *****/
char buffer[128];
for(temp = 0; temp < K; temp++)
{
    //printf("\n\nPacket data before\n");
    //gmp_printf("%Zd\n", i[temp]);

    mpz_export(buffer, &words, 1, 128, 0, 0, i[temp]); //export
    s = write(sockfd, buffer, sizeof(buffer)); //write to stream

    if (s<0)
        printf("Prover: Public Key Write Failed\n");
}
}

```

```

    usleep(200);
}

//printf("\n\nPublic Key:\n");
//for(temp = 0; temp<K; temp++) gmp_printf("%Zd\n\n", i[temp]);
/***** STOP Send Public Key *****/

stop = stampstop(start);

/***** START Send Modulus *****/
memset(buffer, 0, 128);
mpz_export(buffer, &words, 1, 128, 0, 0, n); //export 'n' to a char
array

s = write(sockfd, buffer, sizeof(buffer));
if (s<0)
printf("Prover: Modulus Write Failed\n");
//printf("Modulus: ");
//gmp_printf("%Zd\n", n);
/***** STOP Send Modulus *****/

stop = stampstop(start);

/***** START Send Witness *****/
memset(buffer, 0, 128);
mpz_export(buffer, &words, 1, 128, 0, 0, x); //export 'x' to a char
array
s = write(sockfd, buffer, sizeof(buffer));
if (s<0)
printf("Prover: X Write Failed\n");
//printf("Witness: ");
//gmp_printf("%Zd\n", x);
/***** STOP Send Witness *****/

stop = stampstop(start);

/***** START Get Challenge Bitmask *****/
char rxbuffer[K];
unsigned long total = 0x0;
int i = 0;

//mpz_t total;          /* response */
//mpz_init(total);
total = 0x0;

s = read(sockfd, rxbuffer, K);
if (s<0)
    printf("Prover: Challenge Read Failed\n");

for(i; i < K; i++)
{
    if(rxbuffer[i])

```

```

        total |= (0x1 << i);
    }

    //printf("%d\n", total);
    /***** STOP Get Challenge Bitmask *****/

    stop = stampstop(start);

    /***** START Send Response Y *****/
    response(y, total);          /* Prover sends response to
    Verifier */

    memset(buffer, 0, 128);
    mpz_export(buffer, &words, 1, 128, 0, 0, y); //export 'y' to a char
array
    s = write(sockfd, buffer, sizeof(buffer));
    if (s<0)
    printf("Prover: Write Failed for Y\n");

    //gmp_printf("Response x      : %Zd\n\n", x);
    //gmp_printf("Response y      : %Zd\n\n", y);
    //printf("total      : %d\n", total);

    stop = stampstop(start);
    usleep(1000000);              //let local clock advance to ensure
randomness

} // END FOR LOOP FOR T

mpz_clear(x);
mpz_clear(y);
mpz_clear(rndseed);

close(sockfd);

return (0);
}
/***** STOP Send Response Y *****/

/*
Set the random seed from /dev/random
*/
void setrndseed()
{
    FILE *rnd;
    mpz_t rndtmp;
    unsigned long int idx;
    time_t t1;

    if (!fastseed) {
        mpz_init(rndtmp);

```

```

    rnd = fopen("/dev/random", "r");

    for (idx = 0; idx < 128; idx++) {
        mpz_set_ui(rndtmp, (unsigned long int) getc(rnd));
        mpz_mul_2exp(rndtmp, rndtmp, idx * 8);      /* left shift */
        mpz_add(rndseed, rndseed, rndtmp);
    }

    fclose(rnd);
    //mpz_clear(rndtmp);
}

else {
    /* Set a faster seed. Do not use this for cryptographic purposes!
*/
    mpz_set_ui(rndseed, (unsigned long int) time(&t1));
    mpz_mul_ui(rndseed, rndseed, (unsigned long int) getpid());
    mpz_mul_ui(rndseed, rndseed, (unsigned long int) getppid());
}
}

//Modified from www.codealias.info 15 July 2009
uint32_t stampstart()
{
    struct timeval tv;
    struct timezone tz;
    struct tm *tm;
    uint32_t start;

    gettimeofday(&tv, &tz);
    tm = localtime(&tv.tv_sec);

    start = tm->tm_hour * 3600 * 1000 + tm->tm_min * 60 * 1000 + tm-
>tm_sec
        * 1000 + tv.tv_usec / 1000;

    return (start);
}

uint32_t stampstop(uint32_t start)
{
    struct timeval tv;
    struct timezone tz;
    struct tm *tm;
    uint32_t stop;
    uint32_t elapsed;

    gettimeofday(&tv, &tz);
    tm = localtime(&tv.tv_sec);

```

```
    stop = tm->tm_hour * 3600 * 1000 + tm->tm_min * 60 * 1000 + tm->tm_sec  
          * 1000 + tv.tv_usec / 1000;  
  
    elapsed = stop - start;  
    printf("Time Elapsed: %d ms\n", elapsed);  
    usleep(100);  
}
```



## **Appendix B: ZKP Verifier Code**

### **ZKP\_VERIFIER/Makefile**

```
# Makefile for the FFS Zero-Knowledge Identification Scheme
implementation
```

```
CC      = gcc
LIBS    = /usr/local/lib/libgmp.a
```

```
zkp: zkp.o verifier.o
    $(CC) zkp.c verifier.c -o zkp -lm $(LIBS)
```

```
clean:
    clear; rm -f zkp *~ *.o core
```

## **ZKP\_VERIFIER/zkp.h**

```
#define TRUE 1
#define FALSE 0

/* Multiplicity of challenge */
#define K 10
/* Number of rounds of the protocol */
#define T 1

mpz_t n;          /* modulus (a Blum integer) */
mpz_t i[K];       /* Prover's public key */
mpz_t rndseed;

void setrndseed();
```

## **ZKP\_VERIFIER/verifier.c**

```
#include <stdlib.h>
#include <stdio.h>
#include <gmp.h>
#include "zkp.h"

/*
  Sends a random bit vector as the challenge (step 2 of the protocol)
*/
unsigned int challenge()
{
    //return (bitmask);
}

/*
  Verifies the response from Prover (step 4 of the protocol)
*/
int verify(mpz_t x, mpz_t y, unsigned int e) {
    int index, result = FALSE;
    mpz_t test;
    mpz_init(test);

    mpz_mul(test, y, y);
    for (index = 0; index < K; index++)
    {
        if (e & (0x1 << index)) mpz_mul(test, test, i[index]);
    }
    mpz_mod(test, test, n);

    //gmp_printf("Verification: %Zd\n\n", test);
    if (mpz_cmp(x, test) == 0) result = TRUE;
    mpz_clear(test);

    return (result);
}
```

## **ZKP\_VERIFIER/zkp.c**

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <gmp.h>
#include <math.h>
#include <time.h>
#include <sys/time.h>
#include "zkp.h"

//network includes
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

/*****
*****
Feige-Fiat-Shamir (FFS) Zero Knowledge (ZK) Identification Scheme:
Networked Implementation:

    Original by Daniele Raffo, 25 JUN 2002 - LIX, Ecole Polytechnique
    Expanded by Nathaniel Rowe, 11 JAN 2012 - AFRL Rome

This networked implementation uses the same FFS ZK format, but is
setup
    to function over a TCP/IP based network.  ZK is preserved.

This program is the ***SERVER / VERIFIER*** edition
Version Information: Version 1.1.3
Version Details:
    Successful ZK authentication between two networked machines.
Fixed
    a bug that causes intermittent bignum errors. Fixed a network
hang
    bug and now allows continually authentication at the Verifier.
Also
    now removing key information from memory for additional security.
Future Release:
    -Make it easier to adjust size of FFS 'K' vector
    -Check use of 'T' vector for inconsistencies
    -Import secret keys from txt files so they are not transmitted
      at the start of each exchange.

*****/
```

```

uint32_t    stampstart();
uint32_t    stampstop(uint32_t start);

int fastseed = TRUE;

int main(int argc, char **argv) {

    unsigned int e;        /* random boolean vector (challenge) */
    int proof, s, temp;
    uint32_t start, stop;  //time stamp start and stop
    //rnd seed was here
    char buffer[128];
    mpz_t yt;              /* define */
    mpz_t xt;

    /***** SETUP NETWORK *****/
    int sockfd, newsockfd, portno, clilen, binder;
    struct sockaddr_in serv_addr, cli_addr;
    clilen = sizeof(cli_addr);
    portno = 12303;

    sockfd = socket(AF_INET, SOCK_STREAM, 0);

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port = htons(portno);

    //bind the given port
    binder = bind(sockfd, (struct sockaddr *) &serv_addr,
sizeof(serv_addr));
    if(binder < 0)
        printf("Verifier: Bind Failed");

while(TRUE)                //start loop to continually accept
connections
{
    printf("\nFeige-Fiat-Shamir ZKP Implementation:\n");
    printf("Network Edition, Version 1.1.2\n");
    printf("Verifier waiting for Prover access request...\n");

    listen(sockfd, 1);

    newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
    if(newsockfd < 0)
        printf("Verifier: Socket Accept Failure");
    /***** END NETWORK SETUP *****/

    //START TIME TIMESTAMP:
    start = stampstart();
    stop = stampstop(start); //first timestamp

    int t_var = T;

```

```

    for(t_var; t_var > 0; t_var--) //START 'FOR LOOP' FOR T, CLOSE ~LINE
178 {
    mpz_init(rndseed); //initialize random seed
    mpz_init(xt);
    mpz_init(yt);

    /***** START Get Public Keys *****/
    for(temp = 0; temp < K; temp++)
    {
        s = read(newsockfd, buffer, 128);
        mpz_import(i[temp], 1, 1, 128, 0, 0, buffer); //import 'public
key' from char buf
        memset(buffer, 0, 128);
        usleep(200);
    }
    //printf("\n\nPublic key:\n");
    //for (temp = 0; temp < K; temp++) gmp_printf("%Zd\n\n", i[temp]);
    /***** END Get Public Keys *****/

    stop = stampstop(start); //second timestamp

    /***** START Get Publishing Modulus *****/
    memset(buffer, 0, 128);
    s = read(newsockfd, buffer, 128);
    mpz_import(n, 1, 1, 128, 0, 0, buffer); //import 'n' from char buf
    //gmp_printf("Publishing modulus: %Zd\n\n", n);
    /***** END Get Publishing Modulus *****/

    stop = stampstop(start); //third timestamp

    /***** START Get Witness *****/
    memset(buffer, 0, 128);
    mpz_init(xt);
    s = read(newsockfd, buffer, 128);
    mpz_import(xt, 1, 1, 128, 0, 0, buffer); //import 'x' from char buf
    //gmp_printf("Received Witness: %Zd\n\n", xt);
    /***** END Get Witness *****/

    stop = stampstop(start); //fourth timestamp

    /***** START Send Challenge *****/
    unsigned int index, bit;
    unsigned long bitmask = 0x0;
    char bitvector[K];

    setrndseed();
    srandom((unsigned int) mpz_get_ui(rndseed));

    for (index = 0; index < K; index++)

```

```

{
    bit = (int)(random() % 2);
    if (bit)
    {
        bitvector[index] = 0xFF;
        bitmask |= (0x1 << index);
    }
    else
    {
        bitvector[index] = 0x00;
    }
    //printf("%d", bit);
}
//printf("\nBITMASK: %d\n", bitmask);

//now we send the challenge back to the prover
s = write(newsockfd, bitvector, K);
/***** END Send Challenge *****/

stop = stampstop(start); //fifth timestamp

/***** START Get Response *****/
memset(buffer, 0, 128);
s = read(newsockfd, buffer, sizeof(buffer)); //get response from
prover

mpz_init(yt);          /* initialize */
mpz_import(yt, 1, 1, 128, 0, 0, buffer); //import 'y' from buffer

//gmp_printf("Response xt      : %Zd\n\n", xt);
//gmp_printf("Response yt      : %Zd\n\n", yt);
//printf("total      : %d\n", e);
/***** END Get Response *****/

stop = stampstop(start); //sixth timestamp

/***** VERIFY AUTHENTICATION *****/
proof = verify(xt, yt, bitmask); /* Verifier verifies if
response matches */

if (proof)
    printf("Authentication successful!\n");
else
    printf("Authentication failed!\n");

stop = stampstop(start); //seventh timestamp

} //END FOR LOOP FOR T

mpz_clear(xt);
mpz_clear(yt);

```



```

    mpz_clear(rndseed);

    close(newsockfd);
}

return (0);
}

/***** END VERIFY AUTHENTICATION *****/

/***** Random Seed Determination *****/
//Set the random seed from /dev/random

void setrndseed() {

    FILE *rnd;
    mpz_t rndtmp;
    unsigned long int idx;
    time_t t1;

    if (!fastseed) {
        mpz_init(rndtmp);

        rnd = fopen("/dev/random", "r");

        for (idx = 0; idx < 128; idx++) {
            mpz_set_ui(rndtmp, (unsigned long int) getc(rnd));
            mpz_mul_2exp(rndtmp, rndtmp, idx * 8); /* left shift */
            mpz_add(rndseed, rndseed, rndtmp);
        }

        fclose(rnd);
        mpz_clear(rndtmp);
    }

    else {
        /* Set a faster seed. Do not use this for cryptographic purposes!
*/
        mpz_set_ui(rndseed, (unsigned long int) time(&t1));
        mpz_mul_ui(rndseed, rndseed, (unsigned long int) getpid());
        mpz_mul_ui(rndseed, rndseed, (unsigned long int) getppid());
    }
}

/***** Time Stamping *****/
//Modified from www.codealias.info 15 July 2009
uint32_t stampstart()
{
    struct timeval tv;
    struct timezone tz;
    struct tm *tm;

```

```

uint32_t start;

gettimeofday(&tv, &tz);
tm = localtime(&tv.tv_sec);

start = tm->tm_hour * 3600 * 1000 + tm->tm_min * 60 * 1000 + tm-
>tm_sec
        * 1000 + tv.tv_usec / 1000;

return (start);
}

uint32_t stampstop(uint32_t start)
{
    struct timeval tv;
    struct timezone tz;
    struct tm *tm;
    uint32_t stop;
    uint32_t elapsed;

    gettimeofday(&tv, &tz);
    tm = localtime(&tv.tv_sec);

    stop = tm->tm_hour * 3600 * 1000 + tm->tm_min * 60 * 1000 + tm-
>tm_sec
        * 1000 + tv.tv_usec / 1000;

    elapsed = stop - start;
    printf("Time Elapsed: %d ms\n", elapsed);

    usleep(100);
}

```

## List of Symbols, Abbreviations, and Acronyms

AFRL	Air Force Research Laboratory
COTS	Commercial-Off-The-Shelf
dBi	decibel isotropic
dc	direct current
FFS	Feige-Fiat-Shamir
GMP	GNU Multiple Precision
MANET	Mobile Ad Hoc Network
MHz	Mega-Hertz
RF	Radio Frequency
TCP/IP	Transmission Control Protocol/Internet Protocol
UAV	Unmanned Aerial Vehicle
WEP	Wired Equivalent Privacy
WPA	Wi-Fi Protected Access
WPA PSK	Wi-Fi Protected Access Pre-Shared Key
ZKP	Zero Knowledge Proof